Scalability Study of Parallel Spatial Direct Numerical Simulation Code on IBM SP1 Parallel Supercomputer

Ulf R. Hanebutte[1]
Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA 23681, USA

Ronald D. Joslin
NASA Langley Research Center
Hampton, VA 23681, USA

Mohammad Zubair[1]
International Business Machines Corporation
Thomas J. Watson Research Center
Yorktown Heights, NY 10598, USA

Abstract

The implementation and the performance of a parallel spatial direct numerical simulation (PSDNS) code are reported for the IBM SP1 parallel supercomputer. The spatially evolving disturbances that are associated with laminar-to-turbulent transition in three-dimensional boundary-layer flows are computed with the PSDNS code. By remapping the distributed data structure during the course of the calculation, optimized serial library routines can be utilized that substantially increase the computational performance. Although the remapping incurs a high communication penalty, the parallel efficiency of the code remains above 40 percent for all performed calculations. By using appropriate compile options and optimized library routines, the serial code achieves 52–56 Mflops on a single node of the SP1 (45 percent of theoretical peak performance). The actual performance of the PSDNS code on the SP1 is evaluated with a "real world" simulation that consists of 1.7 million grid points. One time step of this simulation is calculated on eight nodes of the SP1 in the same time as required by a Cray Y/MP supercomputer. The 32-node SP1 is 2.9 times faster than the Cray Y/MP for the same simulation. The scalability information provides estimated computational costs that match the actual costs relative to changes in the number of grid points.

[1]This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681.

1 Introduction

In a recent review article, Fischer and Patera [4] summarize current work in the area of parallel simulation of viscous incompressible flows. In their work, they discuss parallel solution strategies for Poisson, Stokes, and Navier-Stokes problems. Iterative and direct-solution techniques that utilize domain decomposition for structured and unstructured finite-element grids are presented. The coupled approach (based on a full-implicit temporal treatment) and the decoupled approach with a semi-implicit time step are compared for the Navier-Stokes equations. Examples of solutions to turbulent flow problems with either transpose-based fast Fourier transform (FFT) techniques or distributed FFT algorithms are given. These Fourier examples include the first parallel computation of a viscous incompressible flow by Moin and Kim [12] in 1982 on a 64-processors ILLIAC IV. However, a discussion of parallel three-dimensional (3D) spatial direct numerical simulation (DNS) algorithms for laminar-to-turbulent transition (the subject of this paper) is not included in reference [4].

This report can be considered as a follow-up to the work by Joslin and Zubair [9], in which the performance of the parallel spatial direct numerical simulation (PSDNS) code on the the relatively small and slow INTEL iPSC/860 computer was analyzed. The limited local memory of the iPSC/860 seriously restricted the analysis. Because the IBM SP1 is a newer generation of parallel computer with an increased local memory capacity and improved computation and communication performance, comparison of these two machines is useless. However, the general statements given in reference [9] in regard to the scaling of major kernels of the PSDNS code are valid.

2 Performance

2.1 The Parallel Computing Environment

The IBM SP1 [5] scalable parallel computer utilized in the presented performance study consists of 128 processing nodes. Each node is essentially an IBM RS/6000 model 370 workstation with a clock rate of 62.5 MHz. The local memory is 128 Mb, and the processor data and instruction cache is 32 kb each. The individual nodes are connected by a multistage network that consists of high-performance switches (50μsec latency, 8.5 Mb bandwidth); each switch can support up to 16 nodes. The peak performance obtained by performing one multiplication and one addition on 64-bit floating point numbers per clock cycle is 125 Mflops for each processing node. However, in practice, a FORTRAN code delivers 15–75 Mflops. Although the next-generation parallel computer from IBM, called the SP2 [8, 13], is identical to the SP1 architecture, its node performance has more than doubled and the communication network bandwidth has increased fourfold. For the SP2, the increase in communication bandwidth relative to the computing performance will provide a better balanced system, which should further improve the performance results of the presented code. The access to Argonne's SP1 is controlled by a scheduler, which ensures that the requested node partition is operated in a dedicated mode. Thus, only the user application and some necessary Unix demons are executed on the assigned processor partition.

2.2 The Parallel Application

The PSDNS code developed by Joslin and Zubair [9] has been ported to the SP1 with only minor changes. The original parallel code is based on the message-passing paradigm with explicit data

distribution, which enables good portability among a broad class of parallel computers. The interested reader is referred to references [9] and [10] for algorithmic details of the spatial DNS code. In the PSDNS code, the data are distributed among the n_p processors in block form with a z-mapping. That is, the 3D data are partitioned into n_p blocks that contain n_z/n_p two-dimensional (2D) planes of $n_x n_y$ data items each (Figure 1). To perform local FFT's in the spanwise direction n_z, the data must be remapped. As indicated in Figure 2, an x-mapping allows the utilization of optimized serial FFT library routines [7] in the z direction. The INTEL implementation of the PSDNS code relies on the *xor* algorithm [2] for the global data exchange; the IBM implementation makes use of a global index routine provided by the AIX-parallel environment [6]. As shown in the study by Joslin and Zubair, a significant performance gain can be achieved by utilizing a machine-specific basic linear algebra subprogram (BLAS) level 3 routine [3] for the matrix by matrix multiplication. Because this routine is also available on the IBM as part of the ESSL library [7], the advantage can also be taken in the present implementation. The performance of the application code is further improved through appropriate selection of the compile options. As a result, the run time of the serial code can be reduced by a factor of 2.3 compared with a compilation without any options. For a small test problem (for which Joslin and Zubair [9] obtained 189 Mflops on the Cray Y/MP and 5 Mflops on a single node of the iPSC/860), a single node of the SP1 delivers 52.5 Mflops for the double-precision (i.e., 64-bit) computation.

2.3 Performance Study

To document the performance of the simulation code for a wide range of problem sizes and number of processing nodes, three test suites are considered here. Further, a scaling analysis is presented in which each of the three problem dimensions are scaled individually and the number of processors is kept constant. The performance study is concluded with a discussion of a real-world large-scale simulation. Although thousands of time steps are required for a single simulation performance figures for only one time step of the PSDNS code are presented here. Performance figures for one time step are sufficient because the workload for each time step is constant throughout the simulation.

For all three test suites, performance data are collected for the serial code on a single node of the SP1 and for the parallel code on up to 64 processing nodes. The chosen problem dimensions are representative of actual simulations that are currently performed on Cray-class supercomputers. The wall-normal dimension is fixed at 41 grid points for all three test suites. The number of grid points in the streamwise direction determines whether the cases are considered to be small, medium, or large. For small problems, the streamwise direction consists of 64 grid points; the medium and large suites have 128 and 256 grid points, respectively. For each test suite, the spanwise dimension is varied from 8 to 128 Fourier modes in powers of 2. Thus, experimental performance data can be obtained for problems that range from as small as 64 streamwise, 41 wall-normal, and 8 spanwise grid points (20 992 grid points) to a problem that is 64 times larger and contains 256 streamwise, 41 wall-normal, and 128 spanwise grid points.

The PSDNS code is instrumented with a set of timers to record separate performance data for different parts of the computation (the total and four dominating algorithmic kernels) and the communication. These measurements are wall-clock time. By including the idle time that results from the necessary synchronization points of the code in the time data, processor-independent performance figures can be obtained. Processor idle time is discussed below in conjunction with the

large simulation for which the small serial fraction of the PSDNS code is experimentally determined.

In Figures 3(a), 4(a), and 5(a), the computational times for a single time step of the small, medium, and large test suites, respectively, are given in double logarithmic graphs. The associated communication times are given in Figures 3(b), 4(b), and 5(b). The excellent scaling of the code on the SP1 can be observed immediately. However, large communication costs relative to the computation costs are incurred because of the unbalanced architecture of the current SP1 (ie., network performance lags behind compute performance of processing nodes) on the one hand and the algorithmic communication penalty on the other hand. The communication penalty must be incurred in order to utilize highly optimized serial FFT routines in the spanwise direction. The good scaling of the communication cost with respect to the number of processors is noteworthy because the communication that occurs in the PSDNS code involves a complete exchange, which represents a stringent test to the communication network.

The speedup of a parallel code for fixed-size problems is an important performance metric. In Figures 6(a), 7(a), and 8(a), the actual speedup of the complete calculation for each case is given. The performance of the algorithm can be improved by scaling the problem size by either increasing the number of spanwise grid points or increasing the number of streamwise grid points. However, the code is less sensitive to changes in the size of the streamwise dimension than it is to changes in the number of spanwise grid points. For all test cases, the parallel efficiency of the PSDNS code stays above 40 percent, even when 64 processing nodes are utilized.

A theoretical speedup metric can be obtained by ignoring all communication costs. These metrics are given in Figures 6(b), 7(b), and 8(b). For large problems with 64 and 128 spanwise grid points, a superlinear speedup is observed. The superlinear theoretical speedup seen for the large problems is not a surprise. The good scalability of the algorithm, combined with the better memory access of the local portion of the distributed data structure is an obvious explanation. For a discussion of superlinear speedup, the reader is referred to reference [14].

To further examine the performance of the simulation algorithm, a cost breakdown and an itemized speedup are given for each of the three test cases with 64 spanwise grid points in Figures 9, 10, and 11. The four dominating kernels of the algorithm, for which the operation count and a normalized count are given in Table 1, are the matrix-matrix multiply (an n_y, n_y matrix is multiplied by an n_y, n_x matrix), the one-dimensional FFT in the spanwise direction, a tridiagonal solver, and a pentadiagonal solver. The cost of each kernel relative to the computational cost shows that both the FFT and the matrix-matrix multiply each require roughly 30 percent of the total computing time. When the number of processors is small, the cost for the FFT routine is higher than for the matrix-matrix multiply. The tridiagonal and pentadiagonal systems remain nearly constant at about 10 and 5 percent, respectively. The cost for communication is relatively high, and for a large number of processors the communication costs are equivalent to 80–90 percent of the computational cost. The slight drop in the communicational costs from 32 to 64 processors for the medium and large test cases can be attributed to the fact that idle time is included in the overall computational cost.

The itemized speedup curves presented in Figures 9(b), 10(b), and 11(b) show the following: a superlinear speedup for the FFT kernel; an ideal linear speedup for the tridiagonal solver; a nearly ideal speedup for the matrix-matrix multiply; and a moderate speedup for the pentadiagonal solver. The overall speedup of the computational fraction of the algorithm is close to that of the matrix-matrix multiply (the kernel that dominates the algorithm). In addition, notice the speedup in communication results with 64 processors in comparison with the 2-processor results.

2.4 Complete Data Exchange

The performance of the complete data exchange portion of the algorithm deserves a closer analysis. The startup latency and bandwidth are the two most important quantities for evaluating a communication network. However, to analyze an application code, obtaining only an experimental bandwidth that includes the startup costs is sufficient. Before a meaningful discussion of the bandwidth achieved per processor can be given, both the actual message volume and the message size must be determined. The regular and throughout the simulation fixed communication pattern results in deterministic values for these quantities. The data-exchange routine is called 51 times during one time step of the algorithm. Thus, the message volume during one time step of the double-precision code (i.e., 8 b per data item) is given by $51 \times 8 \times n_x n_y (n_p - 1)\frac{n_z}{n_p}$. For the small test suite, the message volume is given in Figure 12(a) for 32, 64, and 128 spanwise grid points and up to 32 processing nodes. The message volume quickly approaches its asymptotic value of $51 \times 8 \times n_x n_y n_z$. The message size of each individual message drops rapidly with the number of processors, as can be seen in Figure 12(b). The formula for determining the message size is $8 \times n_x n_y n_z / n_p^2$. To obtain the experimental bandwidth, the message volume must be divided by one-half of the required wall-clock time. The factor of one-half is used because the data must be sent as well as received. The experimental bandwidth achieved per processor is shown in Figure 13 as a function of the message size, which is given on a logarithmic scale. For large messages, the experimental bandwidth is close to the maximum value of the network bandwidth (8.5 Mb/sec). However, the startup latency reduces the observed bandwidth for smaller message sizes. The startup latency prevents the communication from being ideally scalable with the number of processors. A fixed-size problem distributed among a larger set of processors necessitates the exchange of more messages of shorter message size. Hence, network contention does not slow down the data exchange.

2.5 Scaling Analysis of the PSDNS Code

The scalability study is summarized in Figures 14, 15, and 16. The test case with 64 streamwise, 41 wall-normal, and 32 spanwise grid points is used as the pivot point for this study, which is carried out on 16 processors. Figure 14(a) depicts the computational costs for varying the streamwise dimension. The slowdown that occurs when the number of grid points in the streamwise direction is doubled or quadrupled is given in Figure 14(b). The FFT, the matrix-matrix multiply, and the communication cost are slightly superlinear; the overall computation time doubles for 128 grid points and quadruples for 256 points.

A variation in the wall-normal dimension of the problem from 41 to 61 and, finally, to 81 grid points has a greater effect on the costs associated with the simulation. Figure 15(a) gives the computational costs for the individual kernels, and Figure 15(b) plots the slowdown. The normalized count for the matrix-matrix multiply is $O(n_y^2)$ (Table 1), which can readily be seen in the slowdown curve for this major kernel. If n_y is doubled, an execution time that is four times larger results for the matrix-matrix multiply. The pentadiagonal solver also shows a slowdown that is more than linear; the FFT, the tridiagonal solver, and the communication costs scale linearly. The scaling of the overall computational cost follows the dominating kernel (matrix-matrix multiply); thus, it exhibits a slowdown of 2.8 when the wall-normal dimension is doubled.

Figure 16(a) shows the costs are effected when the number of spanwise grid points is increased. A nearly linear scaling of the overall execution time (computation plus communication time) is observed for the range of spanwise dimensions (32 to 128 grid points). A closer look at the individual

slowdown given in Figure 16(b) confirms the normalized count of $O(\log_2 n_z)$ (Table 1) for the FFT kernel. Although all other kernels scale linearly, the FFT kernel causes the computational cost to follow the logarithmic increase of the FFT. We can expect that the overall time will increase at a rate that is greater than linear as the number of spanwise grid points is increased.

2.6 Memory Requirement of the PSDNS Code

The memory requirement of the PDSNS code seriously limited the previous implementation on the INTEL Hypercube [9]; however, the SP1, with 128 Mb of core memory on each node, allows the larger simulations to be carried out. The largest grid that can be calculated on a single node of the SP1 contains 671 744 grid points (128 streamwise, 41 wall-normal, and 128 spanwise points). The executable code for this calculation requires 110 Mb of core memory. A rough estimate for the memory requirement for the serial code then can be given as 170 b per grid point. However, because the memory requirement is a function of both the problem size and the number of processors, no simple relationship between executable code size and problem size can be given. In Figure 17, the memory sizes of the executable code for the large test suite are presented. Due to limited space, only the curves for 8 and 128 spanwise grid points are labeled. The reduction in the executable code size caused by the distributed data structure is clearly visible. The total memory requirement (i.e., the size of the parallel executable code times the number of processors) is larger than the memory needed by the serial code (due to the additional overhead that results from the data remapping routine).

2.7 A Large-Scale Simulation

The nonlinear evolution of a crossflow vortex packet on a swept wing has been computed with the spatial DNS code described by Joslin and Street [11]. Because this study required substantial computational resources (i.e., approximately 125 CPU hours on a Cray-2 with a single processor), it is representative of a large-scale simulation. For the SP1 compatible simulation, a grid with 896 streamwise, 61 wall-normal, and 32 spanwise grid points was used. Thus, the computational grid contains over 1.7 million grid points. The Cray Y/MP performs one time step of this simulation in 54 seconds and delivers 240 Mflops. Therefore, the computational expense of one time step is 12 960 Mflop.

The large core memory of the SP1 allows a problem of the same size to be computed on as few as eight processing nodes. The computational costs of the PSDNS algorithm for 8, 16, and 32 nodes of the SP1 are presented in Figure 18. The dashed line gives the total time required by the algorithm to perform one time step. If we compare these SP1 timings with the times required by a single node of the Cray Y/MP and Cray C-90 (marked with solid squares in the same plot) we see that the PSDNS code is highly competitive with these serial supercomputer performances for as few as 8 and 32 processing nodes of the SP1, respectively.

The actual measured execution time, which includes communication and idle time, is given in Table 2 for 8, 16, and 32 nodes of the SP1. An idealized execution time can be obtained by subtracting those times that each processing node spends idle or in communication. Because the serial part of the algorithm is performed only on the first node, two idealized times must be recorded; one value for the first node and another for the remaining processing nodes. The performance of the PSDNS code (in Mflops), based on the actual time and the idealized time, is given in Table 2. The idealized performance of 55 Mflops per processor is noteworthy. Recall that even though the

peak performance of a single node is 128 Mflops, 15–75 Mflops are generally observed for actual applications. The last column of Table 2 shows the memory requirements of the executable code; these numbers show that the code is far from reaching the local memory limit of 128 Mb.

A performance summary for the communication part of the algorithm is given in Table 3. The presented values for the message volume and the message size are calculated with the formula presented in section 2.4. The measured experimental bandwidth agrees well with the values shown in Figure 13 for the large test case.

By using the idealized execution times for node 1 and for nodes 2–n_p in Table 2 (the idealized execution time excludes all idle time and communication costs), one can determine experimentally the serial and parallel fractions of the PSDNS algorithm. The difference between the two execution times is the time spent in the serial part of the parallel algorithm. If we multiply the execution time of node 2 by the number of processors, we obtain the execution time of the parallel portion. In this context, total execution time is equal to the time spent by all processing nodes combined. If we normalize the time spent in the serial and parallel portions of the algorithm with the total execution time we obtain the serial fraction s and the parallel fraction p, respectively. Surprisingly, the serial fraction is only 1.4 percent, and the parallel fraction is 98.6 percent of the total. Amdahl's law [1] provides a theoretical speedup that is derived from these two quantities:

$$S_p = \frac{1}{s + \frac{p}{n_p}}$$

For 8, 16, and 32 processing nodes, the theoretical speedup S_p of the PSDNS code is 7.29, 13.22, and 22.32, respectively. In the limit of $n_p \rightarrow \infty$, the speedup asymptotically reaches the value $1/s$. Even though the parallel granularity of the PSDNS code is restricted for this problem to 32 processing nodes, the theoretical maximum speedup is 71.

3 Conclusions

The expectations raised in reference [9] for the performance of the PSDNS code on a larger and more powerful distributed memory machine may be realized with its implementation on the SP1. In reference [9], due to hardware limitations, only a vague estimate for the performance of a large-scale simulation on a 32-node INTEL iPSC/860 with sufficient core memory was given. Joslin and Zubair concluded that the execution time for the PSDNS code on the 32-node iPSC/860 would be twice the time required by a Cray supercomputer. In this work, we have shown that only eight nodes of the more powerful SP1 are needed to perform such a large-scale simulation in the same amount of time as required by a Cray Y/MP. Furthermore, the utilization of 32 processing nodes on the SP1 reduces the execution time to roughly one-third. Both the parallel efficiency of the PSDNS code (above 40 percent for all performed calculations) on the SP1 and the high serial performance of 52–56 Mflops on a single SP1 node (45 percent of theoretical peak performance) contribute to this success. On 32 processing nodes of the SP1, the PSDNS code is also highly competitive in comparison with the advanced Cray C-90 on large-scale simulations.

The scalability of the computation and communication parts of the PSDNS code have been documented for three test suites. A performance gain can be realized with a more balanced architecture, in which the network bandwidth is increased relative to the processing performance of the nodes. The next generation of IBM's parallel computer, the SP2, which has been recently

introduced, has a greater balance between communication and computational performance. Thus, a higher parallel efficiency can be expected for the PSDNS code on the SP2, in addition to the higher serial performance.

The scalability information obtained by independently varying the number of grid points in each of the three problem dimensions confirms the theoretical scaling analysis and is in agreement with the results obtained on the iPSC/860 [9]. The actual time required for the large simulation on 16 processors can be predicted correctly. In Figure 14(a), the total time for a problem of size $n_x = 64$, $n_y = 41$, $n_z = 32$ is 1.3 sec. The scaling coefficients for n_x can be determined from Figure 14(b) to be 14.3. For scaling the wall-normal dimension, Figure 15(b) gives a factor of 1.6. Therefore, the estimated time for the large simulation is $14.3 \times 1.6 \times 1.3$ sec $= 29.74$ sec, which is close to the value recorded in Table 2. Estimates for both 8 and 32 processors can also be obtained with the speedup information provided in Figure 8(a) for the large test suite. The obtained execution times are 56 and 17 sec, respectively, which again are reasonably close to the actual measured time.

Acknowledgments

The authors gratefully acknowledge use of the Argonne National Laboratory High-Performance Computing Research Facility (HPCRF). The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing.

References

[1] Amdahl, G. (1967). *Validity of the single-processor approach to achieving large scale computing capabilities.* Proceedings of the AFIPS Conference, 483–485.

[2] Bokhari, S.H. (1991). *Complete Exchange on the iPSC/860.* ICASE Report No. 91-4.

[3] Dongarra, J.J., DuCroz, J., Hammarling, S., and Duff, I. (1990). A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, **16**(1), 18–28.

[4] Fischer, P.F., and Patera, A.T. (1994). Parallel simulation of viscous incompressible flows. *Annu. Rev. Fluid. Mech.*, **26**, 483–527.

[5] Gropp, W., Lusk, E., and Pieper, S.C. (1994). *Users Guide for the Argonne National Laboratory IBM SP1.* Argonne National Laboratory.

[6] IBM Parallel Programming Reference, *AIX Parallel Environment, Release 1.0.* SH26-7228-00, (1993).

[7] IBM Guide and Reference, *Engineering and Scientific Subroutine Library, Version 2.* SH23-0526-00, (1992).

[8] IBM Press Release, *Cornell Theory Center First To Receive IBM's Newest High Performance Power Parallel System.* Obtained via World Wide Web, April 5, 1994.

[9] Joslin, R.D., and Zubair, M. (1993). *Parallel Spatial Direct Numerical Simulations on the Intel iPSC/860 Hypercube.* ICASE Report No. 93-53.

[10] Joslin, R.D., Street, C., and Chang, C.-L. (1993). Spatial DNS of Boundary-Layer Transition Mechanisms: Validation of PSE Theory. *Theor. and Comp. Fluid Dyn.* **4**(6), 271–288.

[11] Joslin, R.D., and Street, C. (1994). The Role of Stationary Crossflow Vortices in Boundary-Layer Transition on Swept Wings. Accepted for publication in *Phys. Fluids A.*

[12] Moin, P., and Kim, J. (1982). Numerical investigation of turbulent channel flow. *J. Fluid Mech.* **118**, 341–377.

[13] Saini, S. (1994). *The IBM SP2: Hardware, Software, Porting and Optimization Overiew.* Numerical Aerodynamics Simulation Program, NASA Ames Research Center, NAS User Seminar, July 27, 1994.

[14] Sun, X.-H., and Zhu, J. (1994). *Shared Virtual Memory and Generalized Speedup.* ICASE Report No. 94-2. Also to appear in the Proceedings of the International Parallel Processing Symposium, 1994.

Table 1: Operation Counts for Major Kernels

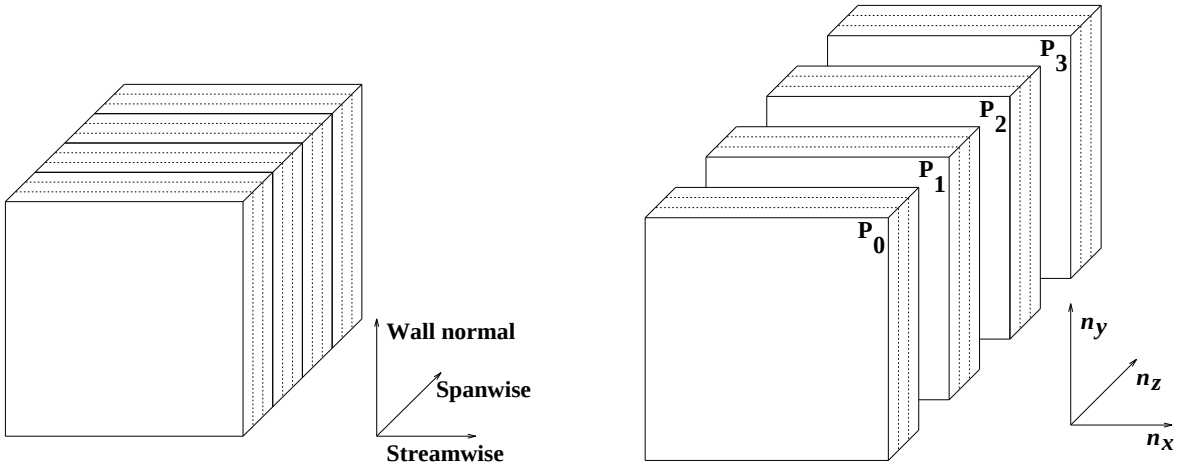
Kernel	Operation count (OC)	Normalization count $= OC/n_x n_y n_z$
MAT-MAT	$O(n_x n_y^3 n_z)$	$O(n_y^2)$
FFT	$O(n_x n_y n_z \log_2 n_z)$	$O(\log_2 n_z)$
TRIDIAG	$O(n_x n_y n_z)$	$O(1)$
PENTADIAG	$O(n_x n_y n_z)$	$O(1)$

Table 2: Performance of large simulation on 8, 16, and 32 Nodes of SP1

Number of processors n_p	Time, sec			Performance, Mflops				Executable code, Mb
	Actual	Idealized node		Actual		Idealized		
		1	2–n_p	Per proc.	Total	Per proc.	Total	
8	53.75	32.4	29.1	30	241	55	440	79
16	29.75	17.7	14.5	27	436	55	880	60
32	18.75	10.2	7.1	22	691	56	1760	50

Table 3: Total Data Exchange for Single Iteration Step of Large Simulation

Number of processors n_p	Comm., sec	Message		Bandwidth	
		Volume, Mb	Size, Mb	Total, Mb/sec	Per Processor, Mb/sec
8	19.0	595	0.208	63	7.9
16	11.0	638	.052	116	7.3
32	7.5	659	.013	176	5.5



(a) Entire domain. (b) Mapped onto four processors.

Figure 1: Computational domain.

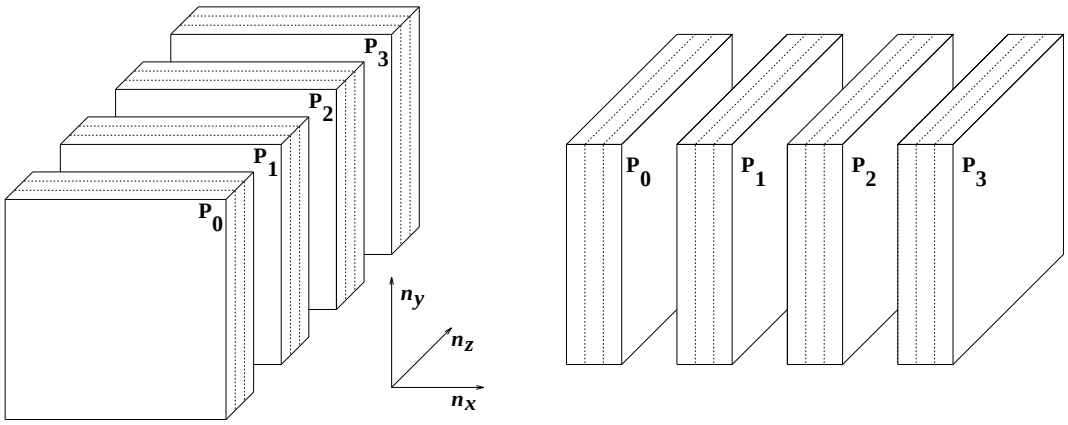
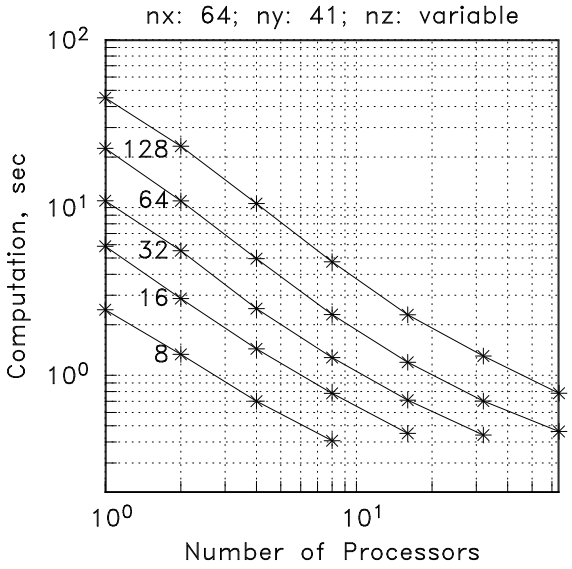
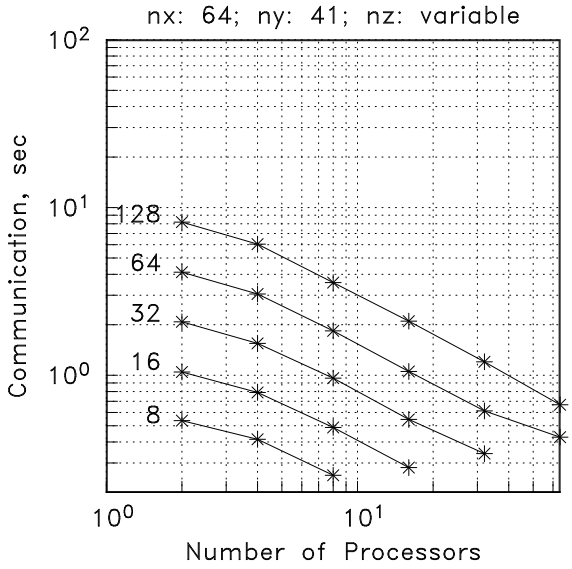


Figure 2: Global remapping results in local FFT's in spanwise direction.

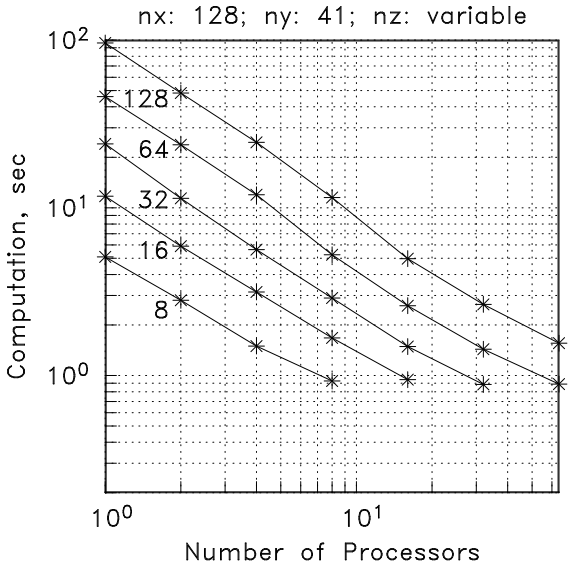


(a) Computational costs.

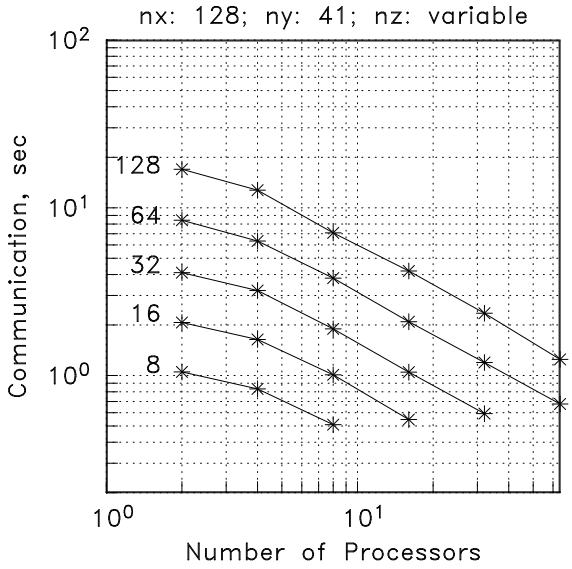


(b) Communication costs.

Figure 3: Computational and communication costs for small test suite.

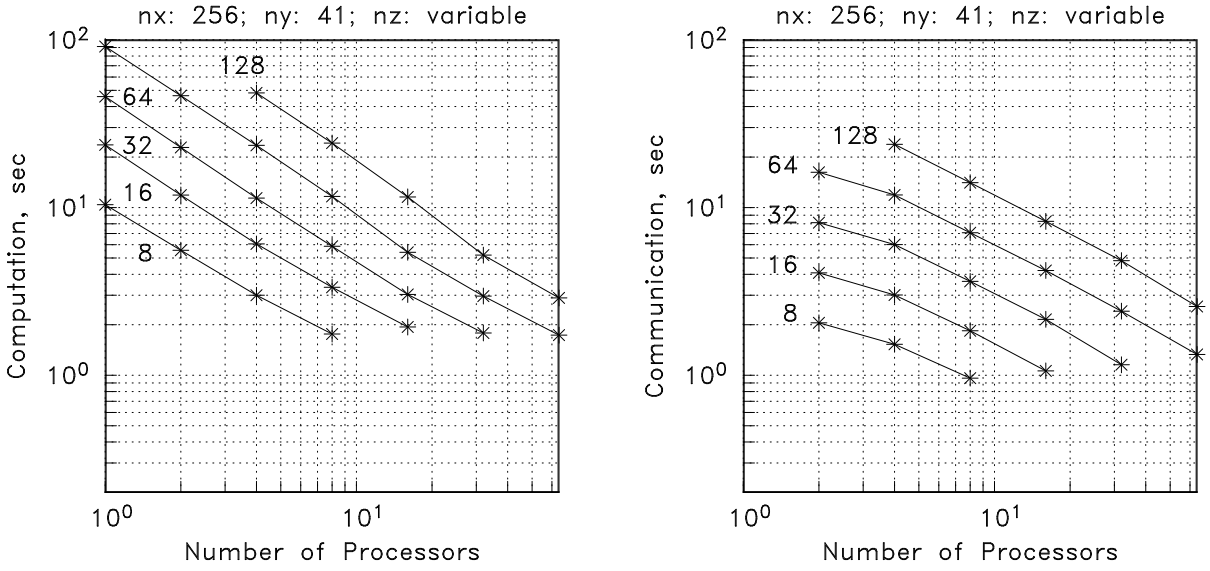


(a) Computational costs.



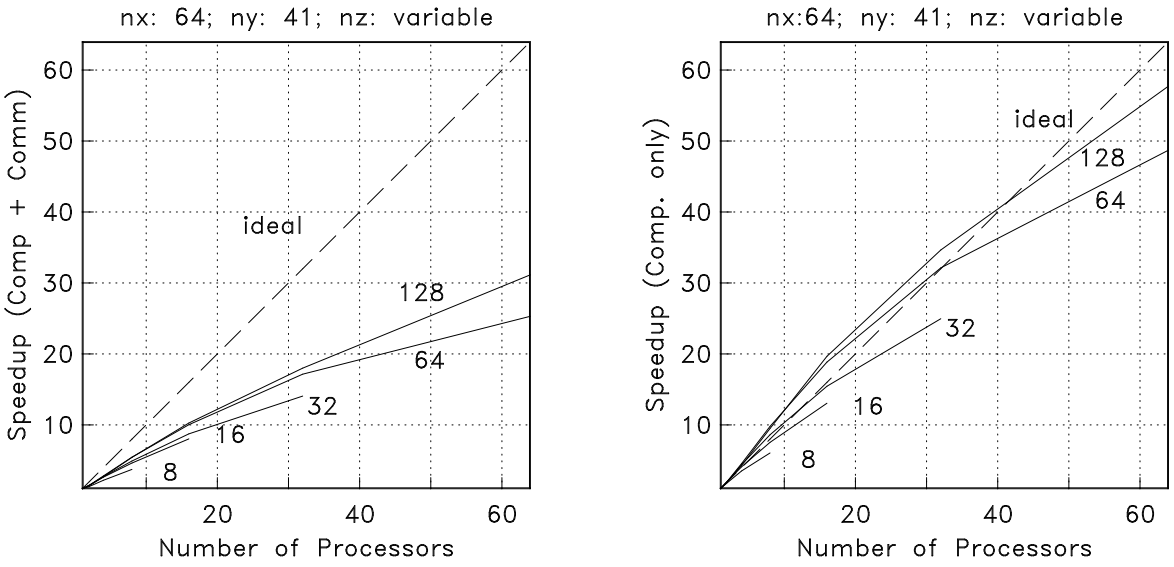
(b) Communication costs.

Figure 4: Computational and communication costs for medium test suite.



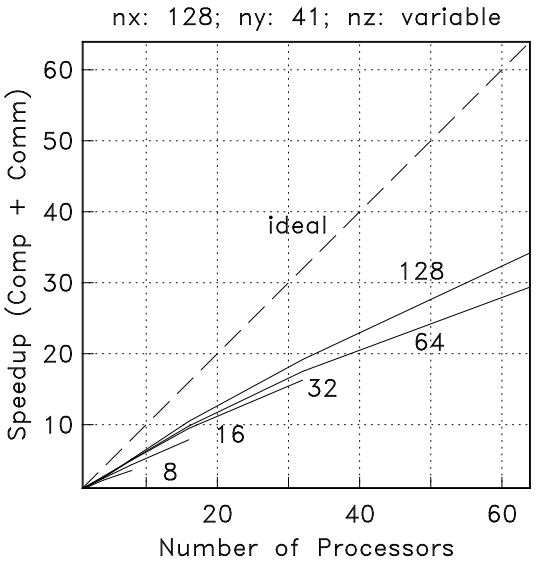
(a) Computational costs. (b) Communication costs.

Figure 5: Computational and communication costs for large test suite.

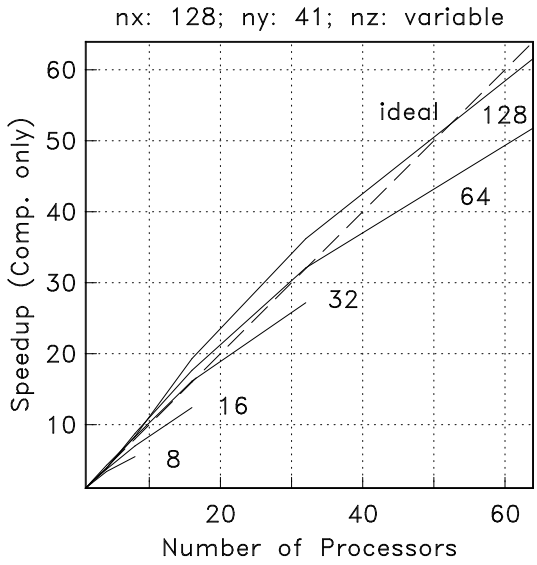


(a) Complete calculation. (b) Computation only.

Figure 6: Speedup for small test suite.

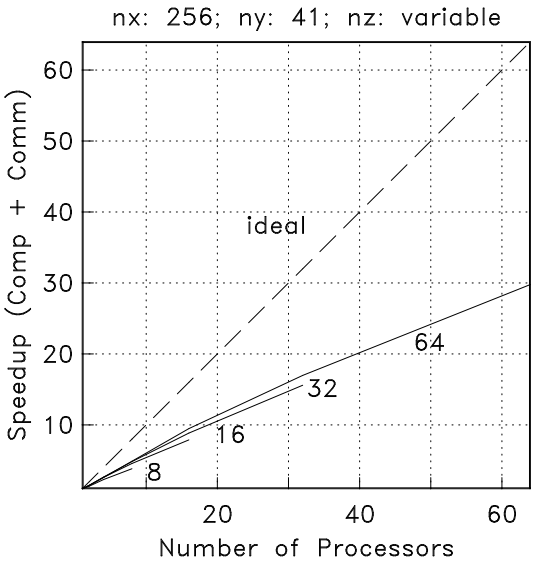


(a) Complete calculation.

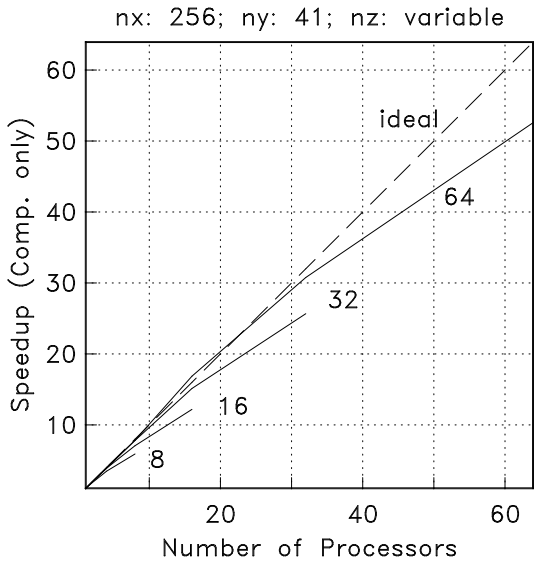


(b) Computation only.

Figure 7: Speedup for medium test suite.

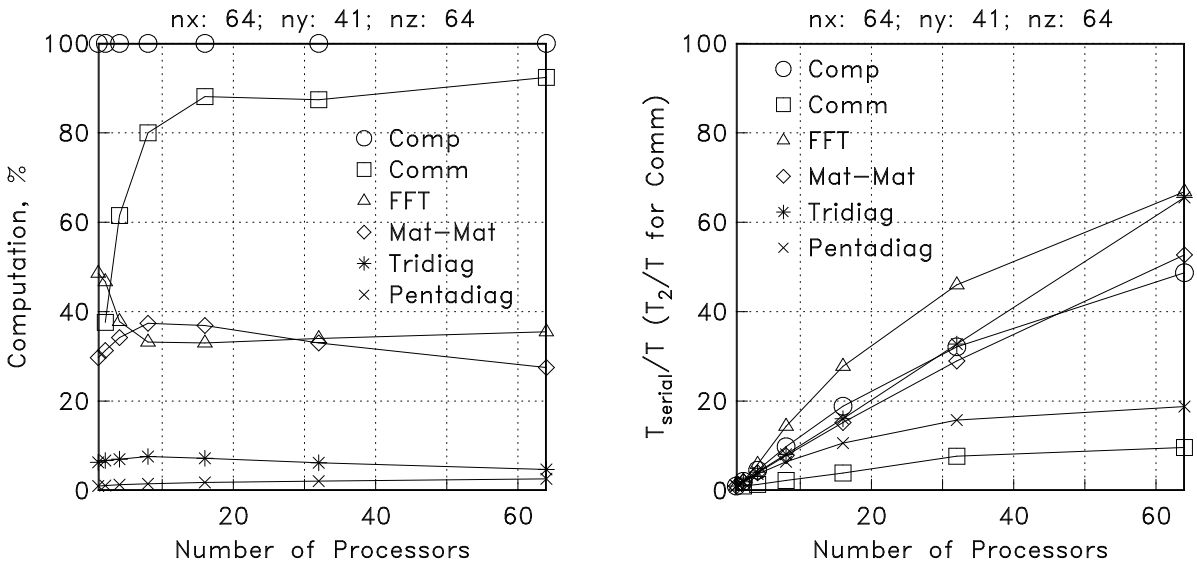


(a) Complete calculation.



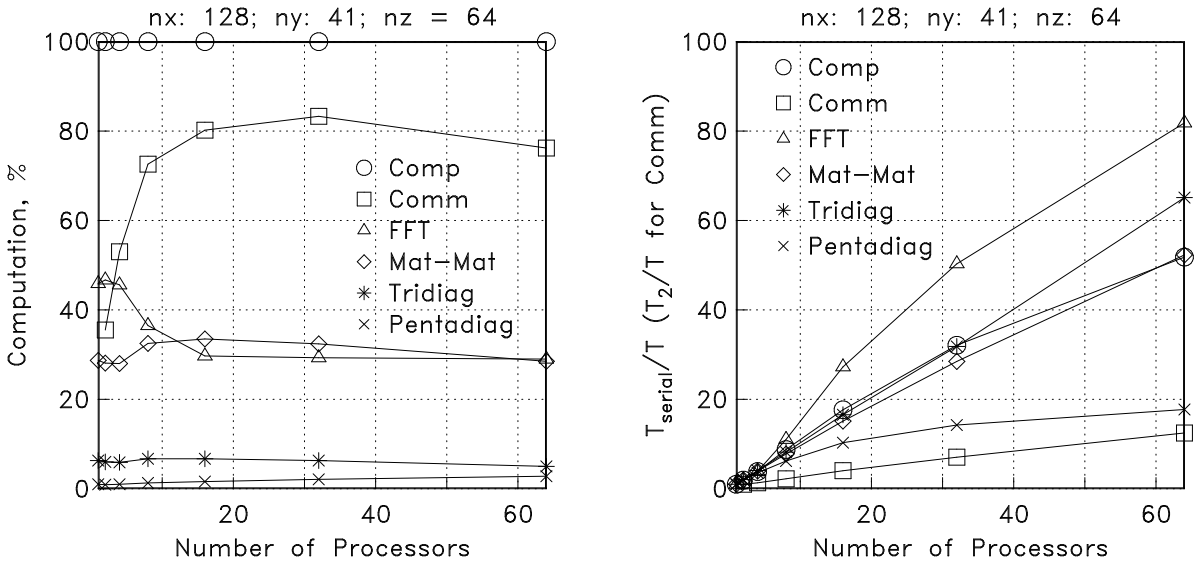
(b) Computation only.

Figure 8: Speedup for large test suite.



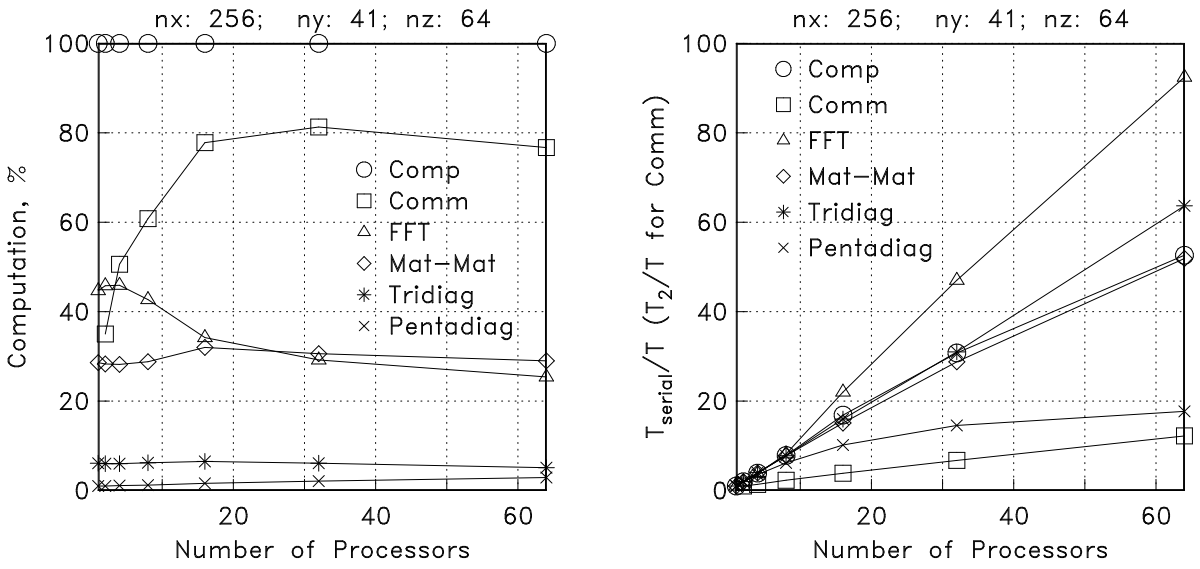
(a) Cost breakdown. (b) Speedup.

Figure 9: Cost breakdown and speedup for small test case with 64 spanwise grid points.



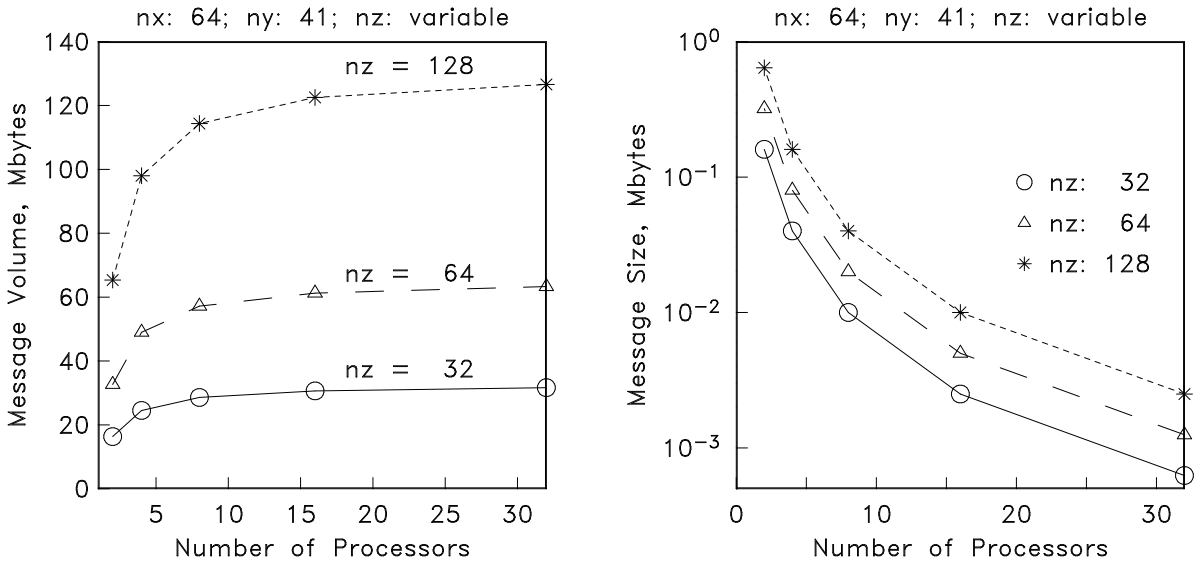
(a) Cost breakdown. (b) speedup.

Figure 10: Cost breakdown and speedup for medium test case with 64 spanwise grid points.



(a) Cost breakdown. (b) speedup.

Figure 11: Cost breakdown and speedup for large test case with 64 spanwise grid points.



(a) Message volume. (b) Message size.

Figure 12: Message volume and size for one time step of small test suite.

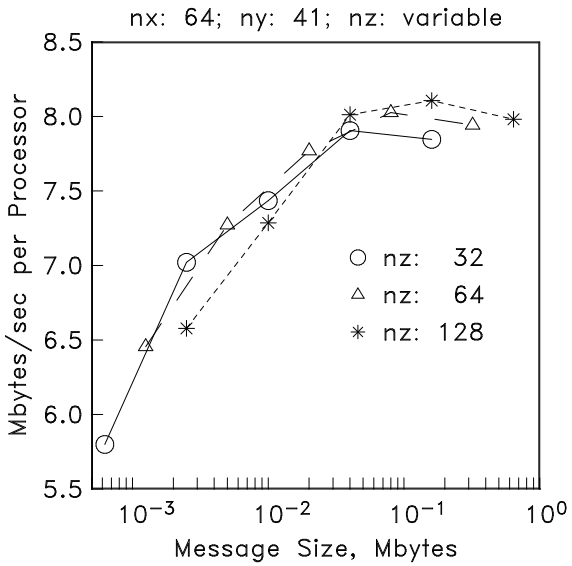
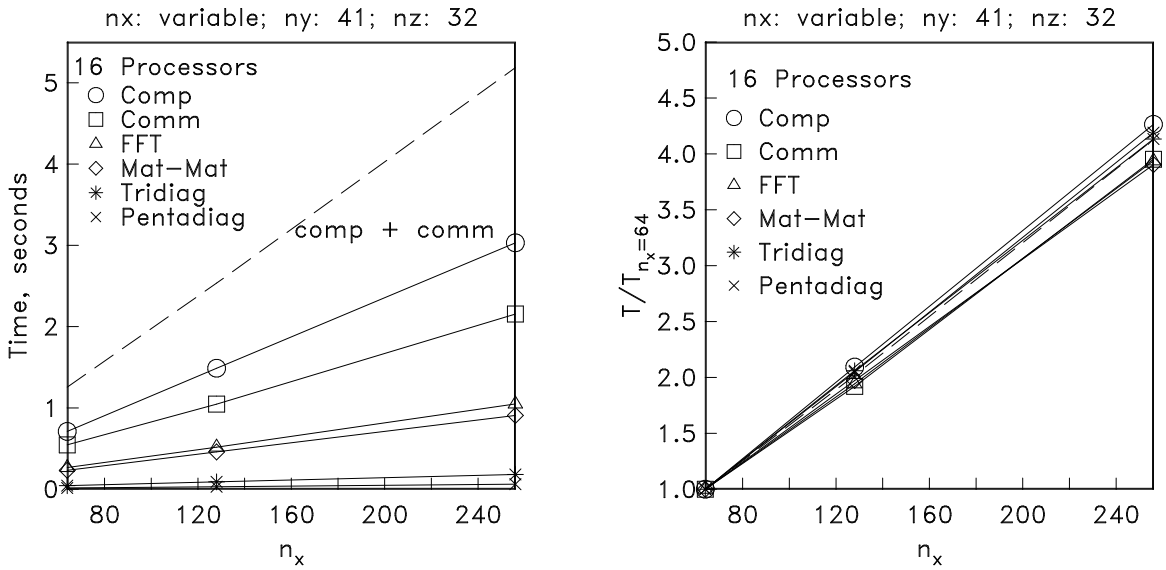


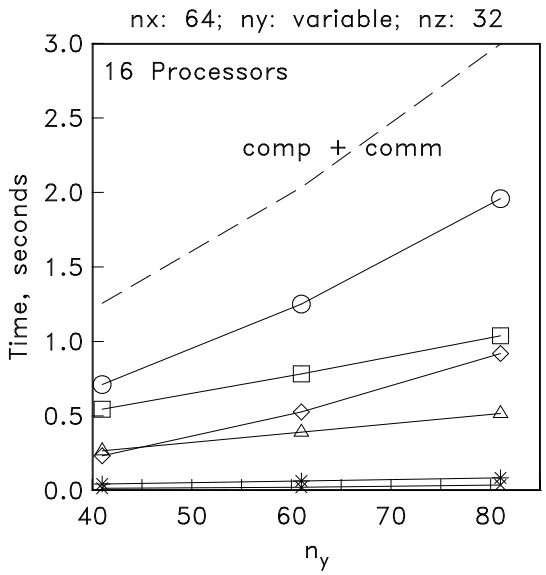
Figure 13: Communication bandwidth per processor as function of message size.



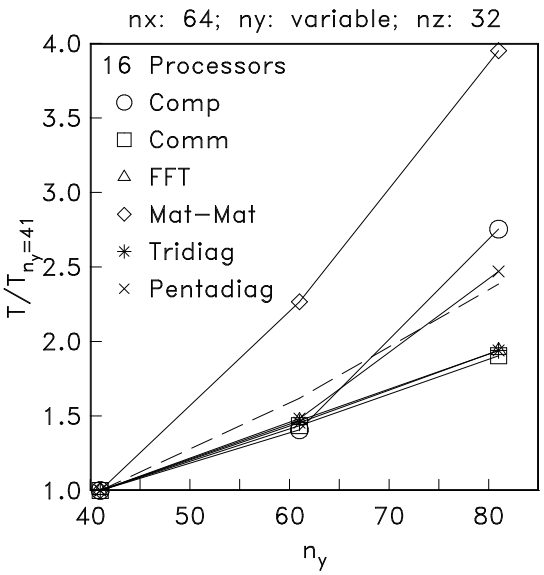
(a) Computational costs.

(b) Slowdown.

Figure 14: Performance of PSDNS code with varying streamwise dimension.

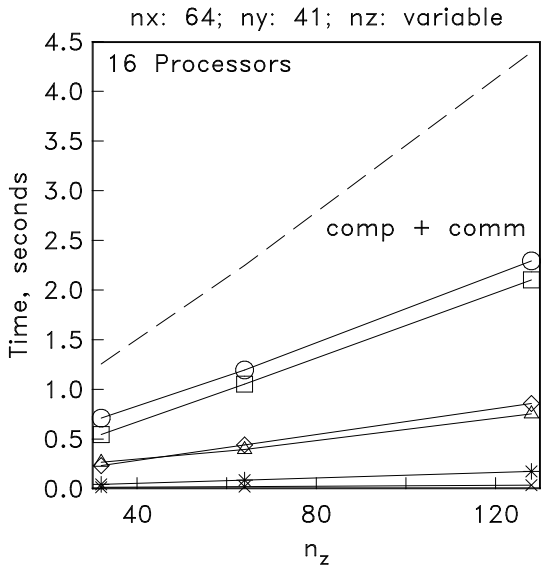


(a) Computational Costs

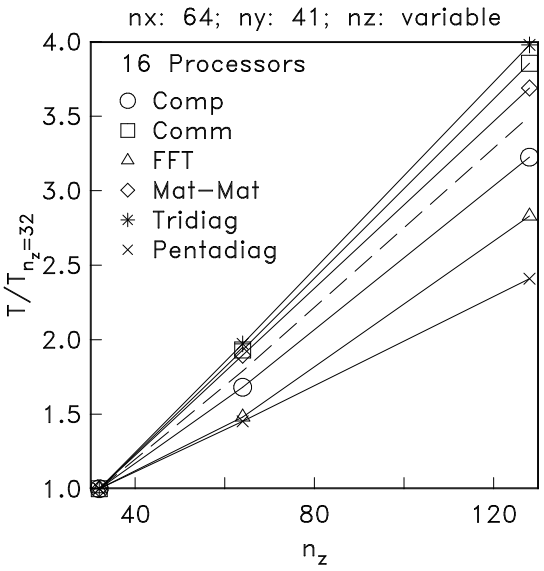


(b) Slowdown

Figure 15: Performance of PSDNS code with varying wall normal dimension.



(a) Computational Costs



(b) Slowdown

Figure 16: Performance of PSDNS code with varying spanwise dimension.

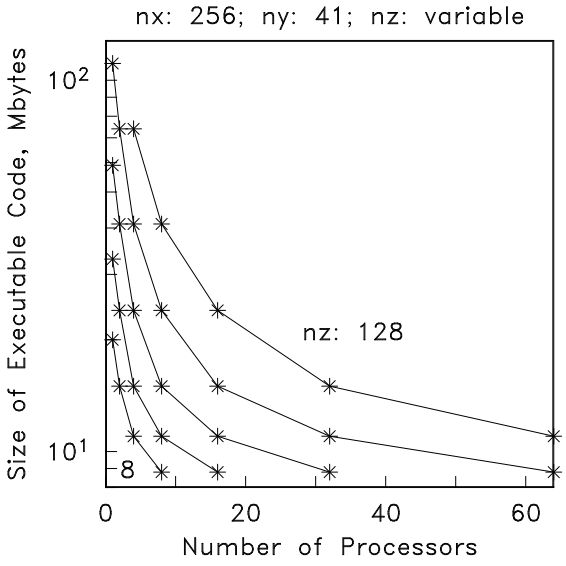


Figure 17: Memory size of executable codes in large test suite.

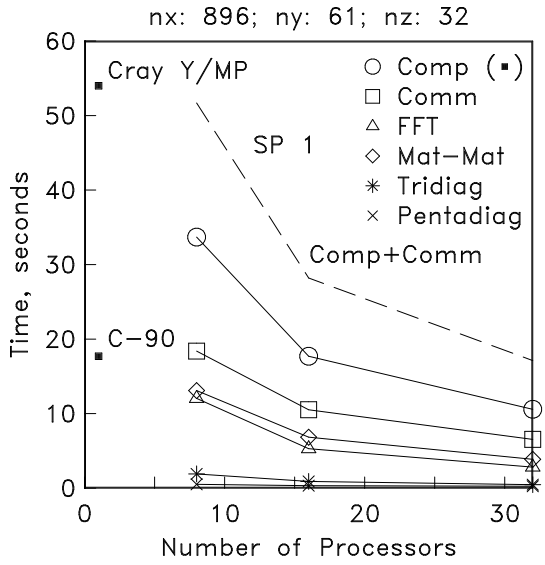


Figure 18: Computational costs for large simulation: IBM SP1 versus Cray Y/MP and Cray C-90.